US 20020010911A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2002/0010911 A1**

Cheng et al. (43) **Pub. Date: Jan. 24, 2002**

(54) **COMPILE TIME POINTER ANALYSIS ALGORITHM STATEMENT OF GOVERNMENT INTEREST**

(76) Inventors: **Ben-Chung Cheng**, Milpitas, CA (US); **Wen-mei Hwu**, Champaign, IL (US)

Correspondence Address:
Steven P. Fallon
GREER, BURNS & CRAIN, LTD.
300 South Wacker Drive, 25th Floor
Chicago, IL 60606 (US)

(21) Appl. No.: **09/770,029**

(22) Filed: **Jan. 25, 2001**

Related U.S. Application Data

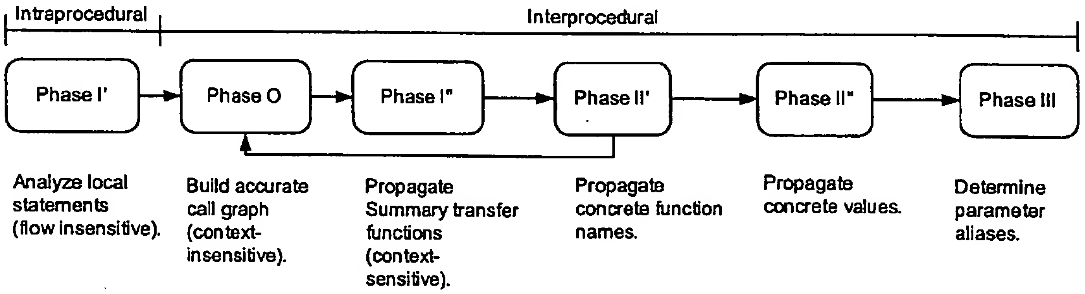
(63) Non-provisional of provisional application No. 60/182,769, filed on Feb. 16, 2000.

Publication Classification

(51) **Int. Cl.⁷** **G06F 9/44**
(52) **U.S. Cl.** **717/4**

(57) **ABSTRACT**

In compiling a program, the present algorithm first analyzes each function in the program as an isolated compilation unit where parameters and global variables are temporarily assumed to have uninitialized values. This stage of the algorithm, the intraprocedural phase, will summarize the intraprocedural behavior of a function in a flow-insensitive manner, including how it can affect memory accesses in the caller and callee functions, and how its memory accesses can be affected by the caller and callee functions. The summarized behavior of each function is the only information to be processed in the next stage, the interprocedural stage. A significant size reduction is achieved in the summarized representation as compared to the full function body. This facilitates aggressive optimization of even large programs.


Intraprocedural
Interprocedural
Phase I'
Phase O
Phase I"
Phase II'
Phase II"
Phase III
Analyze local statements (flow insensitive).
Build accurate call graph (context-insensitive).
Propagate Summary transfer functions (context-sensitive).
Propagate concrete function names.
Propagate concrete values.
Determine parameter aliases.

0089 what
when
where) to propagate

Iteratively propagate
- points-to-relations
- fnc names
- concrete values

US 20030172135A1

(19) **United States**

(12) **Patent Application Publication** (10) **Pub. No.: US 2003/0172135 A1**

Bobick et al. (43) **Pub. Date: Sep. 11, 2003**

(54) **SYSTEM, METHOD, AND DATA STRUCTURE FOR PACKAGING ASSETS FOR PROCESSING AND DISTRIBUTION ON MULTI-TIERED NETWORKS**

(76) Inventors: **Mark Bobick**, Mahopac Falls, NY (US); **Charles P. Pace**, North Chittenden, VT (US); **Paolo R. Pizzorni**, Arlington, TX (US); **Darin S. Deforest**, Phoenix, AZ (US)

Correspondence Address:
KENYON & KENYON
ONE BROADWAY
NEW YORK, NY 10004 (US)

(21) Appl. No.: **09/947,162**

(22) Filed: **Sep. 4, 2001**

Related U.S. Application Data

(60) Provisional application No. 60/229,685, filed on Sep. 1, 2000. Provisional application No. 60/236,864, filed on Sep. 29, 2000. Provisional application No. 60/237,179, filed on Oct. 2, 2000, now abandoned. Provisional application No. 60/254,377, filed on Dec. 8, 2000. Provisional application No. 60/262,288, filed on Jan. 17, 2001.

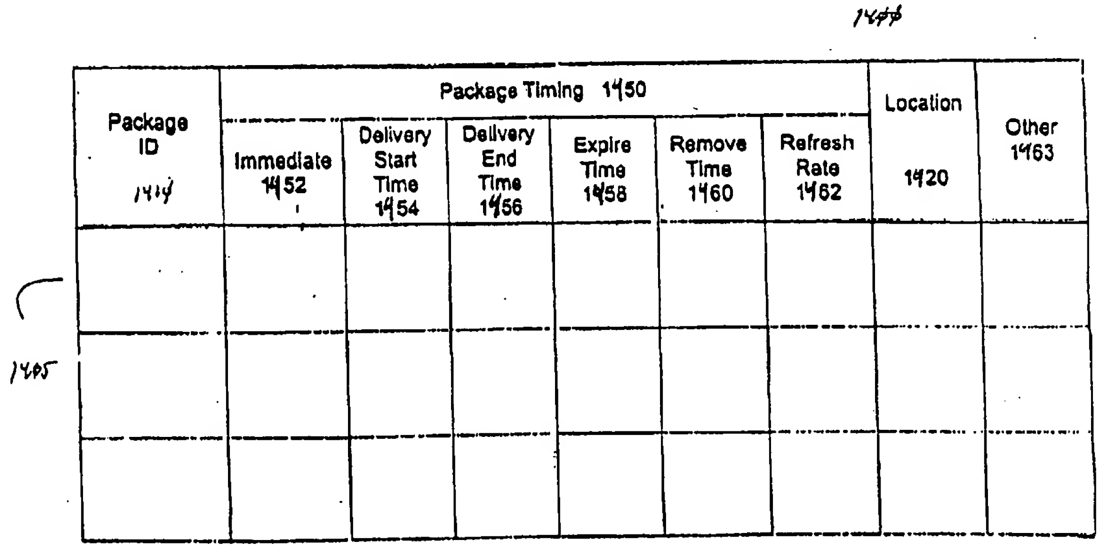
Publication Classification

(51) **Int. Cl.**^{7} **G06F 15/16**; G06F 15/177; G06F 9/44

(52) **U.S. Cl.** **709/220**; 709/201; 709/315

(57) **ABSTRACT**

The present invention provides a system, method, and data structure for packaging assets for processing and distribution over a multi-tiered network. An asset may represent network and/or application components (e.g., data, objects, applications, program modules, etc.) that may be distributed among the various resources of the network. In an embodiment, the package structure includes at least one representation of an asset having a logic/data portion and an asset extended environment portion, and a package extended environment that includes package information associated with at least one asset.



Package ID 1414	Package Timing 1450						Location 1420	Other 1463
	Immediate 1452	Delivery Start Time 1454	Delivery End Time 1456	Expire Time 1458	Remove Time 1460	Refresh Rate 1462		

synchronization asset is propagated

Package Definition Data Structure

US005671419A

United States Patent [19]

Carini et al.

[11] **Patent Number: 5,671,419**

[45] **Date of Patent: Sep. 23, 1997**

[54] **INTERPROCEDURAL DATA-FLOW ANALYSIS THAT SUPPORTS RECURSION WHILE ONLY PERFORMING ONE FLOW-SENSITIVE ANALYSIS OF EACH PROCEDURE**

[75] Inventors: **Paul Robert Carini**, Fairfield County, Conn.; **Michael George Burke**, Westchester County; **Michael James Hind**, Ulster County, both of N.Y.

[73] Assignee: **International Business Machines Corporation**, Armonk, N.Y.

[21] Appl. No.: **490,879**

[22] Filed: **Jun. 15, 1995**

[51] **Int. Cl.**^{6} **G06F 9/44**

[52] **U.S. Cl.** **395/709**

[58] **Field of Search** 395/709

[56] **References Cited**

U.S. PATENT DOCUMENTS

5,428,793	6/1995	Odnert et al.	395/709
5,457,799	10/1995	Srivastava	395/709
5,485,616	1/1996	Burke et al.	395/709
5,535,394	7/1996	Burke et al.	395/709
5,539,907	7/1996	Srivastava	395/705
5,555,417	9/1996	Odnert et al.	395/709

Primary Examiner—Kevin A. Kriess
Assistant Examiner—John I. Chavis
Attorney, Agent, or Firm—Whitham, Curtis, Whitham & McGinn; Louis J. Percello

[57] **ABSTRACT**

A computer implemented method performs flow-sensitive interprocedural data flow analysis without iteration for a class of interprocedural problems. The accuracy of the solution can approach the iterative result without the compile time cost. For interprocedural constant propagation (ICP), this method is more effective than existing methods and costs about the same compilation time. For flow-sensitive ICP over a program call graph (PCG), the method supports recursion while only performing one flow-sensitive analysis of each routine. If the PCG has cycles, a flow-insensitive solution is precomputed for constant propagation. During the flow-sensitive computation, the flow-insensitive result is used for a back edge. This permits a flow-sensitive solution to be obtained in one forward traversal of the PCG. This method can also be used to compute returned constants with one reverse traversal of the PCG. For flow-sensitive USE over a program call graph (PCG), the method supports recursion while only performing one flow-sensitive analysis of each routine. If the PCG has cycles, a flow-insensitive solution for a reference set (REF) is precomputed. During the flow-sensitive USE computation, the flow-insensitive REF solution is used for a back edge. This permits a flow-sensitive USE solution to be obtained in one reverse traversal of the PCG.

5 Claims, 8 Drawing Sheets

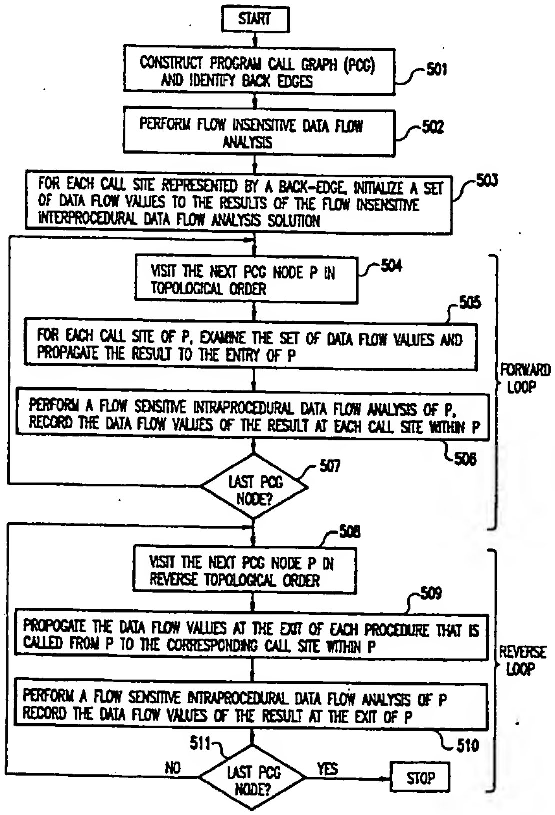

START
CONSTRUCT PROGRAM CALL GRAPH (PCG) AND IDENTIFY BACK EDGES
501
PERFORM FLOW INSENSITIVE DATA FLOW ANALYSIS
502
FOR EACH CALL SITE REPRESENTED BY A BACK-EDGE, INITIALIZE A SET OF DATA FLOW VALUES TO THE RESULTS OF THE FLOW INSENSITIVE INTERPROCEDURAL DATA FLOW ANALYSIS SOLUTION
503
VISIT THE NEXT PCG NODE P IN TOPOLOGICAL ORDER
504
505
FOR EACH CALL SITE OF P, EXAMINE THE SET OF DATA FLOW VALUES AND PROPAGATE THE RESULT TO THE ENTRY OF P
FORWARD LOOP
PERFORM A FLOW SENSITIVE INTRAPROCEDURAL DATA FLOW ANALYSIS OF P, RECORD THE DATA FLOW VALUES OF THE RESULT AT EACH CALL SITE WITHIN P
506
507
LAST PCG NODE?
508
VISIT THE NEXT PCG NODE P IN REVERSE TOPOLOGICAL ORDER
509
PROPOGATE THE DATA FLOW VALUES AT THE EXIT OF EACH PROCEDURE THAT IS CALLED FROM P TO THE CORRESPONDING CALL SITE WITHIN P
REVERSE LOOP
PERFORM A FLOW SENSITIVE INTRAPROCEDURAL DATA FLOW ANALYSIS OF P RECORD THE DATA FLOW VALUES OF THE RESULT AT THE EXIT OF P
511
510
NO
LAST PCG NODE?
YES
STOP

FIG 5 propagate values

US006546551B1

(12) **United States Patent**
Sweeney et al.

(10) **Patent No.: US 6,546,551 B1**
(45) **Date of Patent: Apr. 8, 2003**

(54) **METHOD FOR ACCURATELY EXTRACTING LIBRARY-BASED OBJECT-ORIENTED APPLICATIONS**

(75) Inventors: **Peter Francis Sweeney**, Spring Valley, NY (US); **Frank Tip**, Mount Kisco, NY (US)

(73) Assignee: **International Business Machines Corporation**, Armonk, NY (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/408,224**

(22) Filed: **Sep. 28, 1999**

(51) **Int. Cl.**^{7} **G06F 9/45**

(52) **U.S. Cl.** **717/154**; 717/153; 717/148; 717/165; 717/156

(58) **Field of Search** .... 717/151, 108, 717/116, 131, 132, 133, 154, 155, 153, 56, 157, 128, 109, 113, 104, 148, 165, 156; 707/1, 10, 103 R; 711/171; 345/594

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,241,673	A	*	8/1993	Schelvis	707/103 R
5,794,041	A	*	8/1998	Law et al.	717/104
5,872,973	A	*	2/1999	Mitchell et al.	709/305
5,907,843	A	*	5/1999	Cleron et al.	707/1
5,915,252	A	*	6/1999	Misheski et al.	707/10
5,983,020	A	*	11/1999	Sweeney et al.	706/47
6,093,216	A	*	7/2000	Adl-Tabatabai et al.	717/128
6,230,314	B1	*	5/2001	Sweeney et al.	717/108
6,292,933	B1	*	9/2001	Bahrs et al.	717/107
6,401,182	B1	*	6/2002	Sweeney	711/171
6,442,748	B1	*	8/2002	Bowman-Amuah	717/101

OTHER PUBLICATIONS

Title: Reflection in an Object–Oriented Concurrent Language, author: Watanabe et al, ACM, 1988.*

Title: Cost effective object space management for hardware assisted real time garbage collection, author: Nilson, ACM, 1992.*

Title: Call graph construction in Object–Oriented Languages, Grove et al, ACM, Oct., 1997.*

Title: Constraint systems for useless variable elimination, ACM, Wand et al, Jan., 1999.*

Title: Thinking in Java, author: Bruce Eckel, Publication date: Feb., 1998.*

Title: Re–engineering clan Hierarchies Using Concept Analysis, ACM, Snelting et al, 1998.*

Chuck McManis. *Take an In–Depth look at the Java Reflection API.* wysiwyg://5/http://www.javaworls.com/jw–09–1997/jw–09–indepth.html.

Raja Vallee–Rai, et al. *Soot–a Java Bytecode Optimization Framework.* Sable Research Group, School of Computer Science, McGill University.

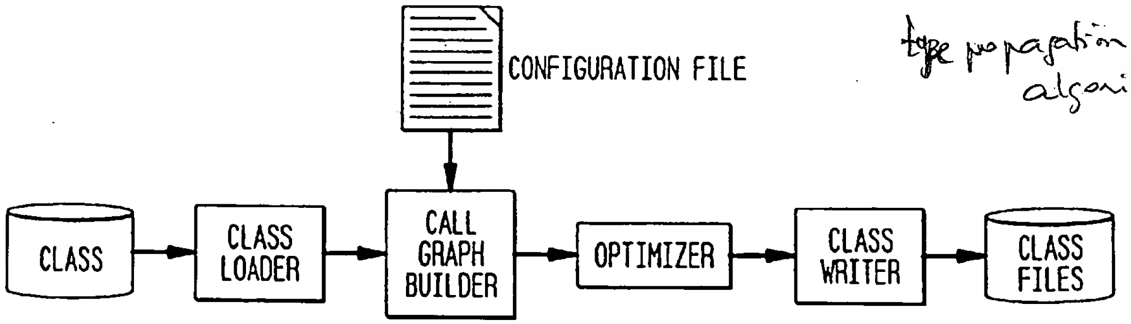
(List continued on next page.)

Primary Examiner—Gregory Morse
Assistant Examiner—Chameli C. Das
(74) *Attorney, Agent, or Firm*—F. Chau & Associates, LLP

(57) **ABSTRACT**

The present invention is capable of accurately extracting multiple applications with respect to a class library. The invention relies on a configuration file for an application program and/or library, which describes how program components in the program/library should be preserved under specified conditions. The invention may be used in application extraction tools, and in tools that aim at enhancing performance using whole-program optimizations. The invention may be used as an optimization to reduce application size by eliminating unreachable methods. In the alternative, the invention may be used as a basis for optimizations that reduce execution time (e.g., by means of call devirtualization), and as a basis for tools for program understanding and debugging.

20 Claims, 4 Drawing Sheets



type propagation-based algorithm